REMARKS

Claims 1-2 and 4-21 are pending in the present application, claim 3 having been cancelled herein. The Office Action and cited references have been considered. Favorable reconsideration is respectfully requested.

Claims 1-21 were rejected under 35 U.S.C. § 102(b) as being anticipated by Collberg (U.S. Patent No. 6 668 325). This rejection is respectfully traversed for the following reasons.

Claim 1 recites a method for securing a computer system which comprises at least a code interpretation module and memory capacities for storing a interpreted code having measurable physical imprints provided from the code interpretation module. In order to make more difficult attacks based on physical measurements or requiring synchronization with the interpreted code, the method consists of introducing at least two types of alternatives in the execution times of the interpreted code, which have an effect on the execution times of the interpreted code or on its measurable physical imprint. The alternatives are introduced according to at least one of the following steps: a first step of causing at certain places of an interpreted code bypasses towards new portions of code which do not belong to the original code in order to complicate the synchronization and the physical imprint of the execution, and/or a second phase of proposing a plurality of implementations of certain instructions, each requiring a different execution time and/or having a different physical imprint and providing an identical result, so that two executions of one of the certain instruction within a same code may be performed by two different implementations. This is not taught, disclosed or made obvious by the prior art of record.

Generally speaking, the main distinctions between Collberg and Applicants' invention stem from the fact that Collberg's aim is to prevent (or at least hamper) the reverse engineering of software (typically for the purpose of protecting intellectual property). In contrast, the method according to Applicants' invention aims at preventing attacks against software (for example to extract cryptographic keys) during their execution. The consequences of this original difference are manifold:

1. Collberg's assumption is that attackers have access to the code of the software (Collberg: column 8, lines 1-6) and can manipulate it (*e.g.*, for reverse engineering purposes) whereas the assumption underlying Applicants' invention is that attackers do not (and should not) have access to the code. Applicants' invention precisely aims at preventing such access.

2. All the embodiments of the method according to Applicants' invention rely on modifications of the observable behaviour of the software (measurable physical imprint) whereas Collberg's objective is opposite: Collberg tries to avoid such variations as much as possible (Collberg: column 2, lines 64-66; column 3, lines 5-16, column 36, lines 21-23, column 37, lines 49-53).

3. Collberg's ultimate objective is to modify the appearance of the code to make it difficult to understand (Collberg: column 6, lines 9-15, column 10, lines 26-30). Applicants' invention is not concerned by the appearance of the code; its only purpose is to modify the behaviour of the code.

4. The transformations in Collberg are selected to minimize execution costs (Collberg: column 5, lines 47-51). All the transformations provided in Applicants' invention increase execution costs (and some strategies may even rely on the deliberate introduction of large execution costs).

5. The method according to Applicants' invention works at the level of code execution and relies on the existence of an interpretation code module, whereas

Collberg operates at the level of code structure and does not require the existence of an interpretation code module. It is true that one embodiment of Collberg is the design of a Java obfuscator and Java happens to be executed by a bytecode interpreter, but the method set forth in Collberg does not rely on the existence of such interpreter (or make specific use of it): it can very well be applied in a different context (*e.g.*, in which software is compiled into native – or machine – code), without any interpretation layer (Collberg: column 2, lines 59-64, column 4, lines 40-44). As a matter of fact, in the perspective of reverse engineering (which is the motivating context for Collberg), the existence of a virtual machine is considered as a handicap (because it makes decompilation easier) rather than an opportunity (Collberg: column 8, lines 13-25).

In light of the aforementioned differences, one can see that Collberg and Applicants' invention concerns complementary rather than overlapping methods. In particular, Applicants' invention protects software during its execution (preventing attackers to guess sensitive data or binary code) but offers no protection whatsoever against reverse engineering. In contrast, Collberg protects software against reverse engineering from attackers who already have access to its binary code (and would like to guess the underlying logic, structure or source code) but offers no protection against attackers spying the code during its execution.

Collberg does not require the existence of an interpretation code module (I 5); its purpose is not to make more difficult attacks based on physical measurements or requiring synchronization with the code: attackers do not need to execute the software, they are assumed to have access to the code (I 1); it is not based on modifications of measurable physical imprints but tries to avoid them as much as possible (I 2). Collberg's objective is to modify the appearance of the code rather than its behaviour (I 3).

The Examiner's opinion appears to be more particularly based on a paragraph of Collberg which discloses a method of controlling a computer so that software running on, stored on, or manipulated by the computer exhibits a predetermined and controlled degree of resistance to reverse engineering, including applying selected obfuscating transformations to select parts of the software in which a level of obfuscation is achieved using a selected obfuscation transformation so as to provide a required degree of resistance to reverse engineering, effectiveness in operation of the software and size of transformed; and updating the software to reflect the obfuscating transformations.

It clearly appears that Collberg's method does not correspond to the method which is claimed in Applicants' amended claim 1. More particularly, Collberg does not disclose or suggest a step of introducing alternatives having an effect on the execution time of the interpreted code or on its measurable physical imprint. As stated in the specification text the claimed method involves the following types of alternatives:

- a first type which consists to cause at certain places of an interpreted code bypasses towards new portions of code (which do not belong to the original code) in order to complicate the synchronization and the physical imprint of the execution, or
- by proposing a plurality of implementations of certain instruction each requiring a different execution time and/or having a different physical imprint and providing an identical result so that two executions of this instruction within a same code may be performed by two different implementations.

Hence, by introducing distortions in the execution times and by modifying the physical effect of the execution, both types of the above alternatives will make any correlation attempt more difficult between the observed physical expressions of an interpreted code and its functionalities. Advantageously this method is able to make the apparent executed code different at each execution and will therefore make the discovery of the actual code of the application more difficult.

Applicants' invention provides a method whereby the correlation between physical measurements (during execution of the program) and characteristics of this program or data operated by this program is rendered more complex, provided that the protected code is not native but consists of byte code. In contrast, the purpose of Collbergs is to increase the degree of resistance to reverse engineering (direct decompilation) by using obfuscating transformation to select part of the software so as to render more complex the meaning or the decompilation of the program from the knowledge of its code (without executing this program).

The adaptation of the obfuscation level is made by using criteria as the cost of the obfuscation transformation. Once again in contrast to Collberg's purpose, Applicants' method is directed against attacks during the execution of the code in order to recover information about operated data or to disturb its execution.

For these reasons, Applicants respectfully submit that claim 1 is patentable over the prior art of record.

Amended claim 2 states that the first alternative in the execution times of the interpreted codes comprises a first mode of introducing bypass codes including bypass instruction in certain particular locations of the interpreted code, said introducing mode being made at each execution upon generating the interpreted code by a code generator.

These characteristics are not disclosed or suggested by Collberg, which, on column 2, lines 64 to 67, states that obfuscation code transformations can be applied to any languages constructs. These transformations are made in a preprocessing step. For this reason, this solution is not able to make the apparently executed code, different at each execution as in the method as claimed in Applicants' claims 1 and 2. Further, the "bypass codes" referred to in claim 2 consist of portions of byte code whereas Collberg, at column 2, lines 58-64, refers to control and data structures at the level of the original code (code transformation).

The characteristics of the original claims 3 have been incorporated into the newly amended claim 1. For this reason this original claims 3 has been cancelled provided that the characteristics of this original claim 3 concern the introduction of several implementations of byte code instructions; Collberg does not propose to introduce several implementations of byte code instructions; it rather works at the level of the code itself and proceeds by transformations of this code (including, possibly, the addition of new instructions, but no modification of their implementation). In addition, (Collberg: column 18 lines 15-27) proposes to include a transformation table to convert specific sections of Java bytecode into a different virtual machine code *to be interpreted by a virtual machine interpreter included within the obfuscated application.* Despite the fact that Collberg is not very precise regarding this specific mechanism, it can be understood from (Collberg: column 18, lines 15-27) that it proposes to introduce one (or several) interpretation layer(s) on the top of the standard Java byte code interpreter rather than modifying the Java byte code interpreter itself (or providing several implementations of specific byte code instructions as in the original claim 3). As set forth in Collberg "there is usually an order of magnitude slow down for each level of interpretation"; this is not the case for the method set forth in the original claim 3 which does not introduce new interpretation levels.

The “bypass codes” referred to in claim 4 consist of one or more instructions specific to certain particular locations of the code, which does not appear in Collberg (see column 3, lines 5-16).

Claims 5 provides that the bypass instructions are associated with security levels. Applicants respectfully submit that Collberg does not disclose bypass instruction as defined in Applicants’ specification and claims. In column 2, lines 26-39, Collberg discloses a method including identifying at least one source code input files corresponding to the source software for the application to be processed, selecting a level of obfuscation, a cost, reading and passing the input files … providing information identifying data types, data structures and control structures, constructing appropriate tables to store this information, preprocessing information about the application in response to the preprocessing step, selecting and applying obfuscating code transformation to source code objects… and outputting the transformed software. This paragraph confirms that the Collberg's method is quite different from that of Applicant: this paragraph does not teach introduction of bypass instructions upon generating code by a code generator which is guided to produce these instructions by annotation in the source code and allowing the designation of sensitive code.

Further, the security levels recited in claim 5 do not appear in Collberg because they quantify the difficulty of achieving security attacks during code execution (such as attacks requiring synchronisation with the code or measurements of its physical imprint) which is not an issue in Collberg (I 3). Collberg’s criteria for applying code transformations (Collberg: column 6, lines 9-24) are related to two other factors: (1) the quality of the transformation to make the code difficult to analyse and understand; (2) the execution penalty.

The introduction of bypass code in the implementation of the interpreter itself is not considered in Collberg. Collberg's method does not work at the interpreter level; actually Collberg does not even rely on the language being interpreted (I 5). (Collberg: column 15, lines 19-42) explicitly refers to the transformation of the code of the source application rather than code of an interpreter executing the application.

Claim 6 recites a second mode for introducing "bypass codes" is used which consists of introducing the bypass code which the interpreter is implemented. The paragraph (column 15 lines 19 to 42) of Collberg cited by the Examiner discloses obfuscating transformations that attempt to obscure the control-flow of the source application. This paragraph does not teach introducing a bypass code while an interpreter in implemented.

Claim 7 proposes an extension of the interpreter such that portions of code can be executed or not depending on decisions (either systematic or random) taken by the interpreter at run time. No similar method is proposed in Collberg which does not work at the interpreter level and does not aim at modifying the physical imprint or observable behaviour of the code. (Collberg: column 2 lines 22-39) does not refer to the interpreter and does not apply at run time: it refers to different choices which are available during the process of transforming the code (thus prior to any code execution).

With regard to claim 8, the paragraph (column 18 lines 54-57) of Collberg cited by the Examiner is directed to an automatic parallelization of a program, not to increase the performance of applications, as usual, but to obscure the actual flow of control. Collberg gives two solutions in this respect:

1) create dummy processes that perform no useful task; and

2) split a sequential section of the application code into multiple sections executing in parallel.

Collberg, at column 18, lines 54-57, proposes to parallelize a program in order to obscure the flow of control. Again, it refers to a static transformation; it does not propose, as in claim 8, the introduction of bypass code performing calculation depending on data known at execution time.

With regard to claim 9, the word "random" in the cited paragraph of Collberg: (column 15, lines 32-25) refers to decisions taken at "transformation time" (by an automatic transformer or by a human supervisor) rather than at run time as in claims 8 and 9. In other words, in Collberg, after the decisions have been taken (possibly including random transformations), the program will always behave in the same way, as opposed to the method of claim 9 in which programs behave at each run in a different way because the (possibly random) decisions are taken by the interpreter at run time.

In the paragraph 15 lines 32-35 Collberg states that control ordering transformations randomize the order in which carried out. Therefore such a randomization is not used to obtain the same effect as the applicant's one (random draw of an extra datum during the execution of a superfluous calculation). For this reason the subject matter of claim 9 is not anticipated by Collberg.

With regard to claim 10, in the paragraph at column 10, lines 65-67, Collberg states that the user also selects the required level of obfuscation (eg, potency) and the maximum execution time/space penalty that the obfuscator is allowed to add to the application. In contrast with the Examiner's assertion, this paragraph does not concern an improvement of a mode for realizing a bypass code and does not teach a step of attaching different security levels to the implementation of instructions and associating them with all the more complex implementations.

With regard to claim 11, in the paragraph at column 15, lines 26-43, Collberg states that for transformations that alter the flow of control, a certain amount of computational overhead will be unavoidable. For Alice this means that she may have to choose between a highly efficient program, and one that is highly obfuscated. An obfuscator can assist her in this trade-off by allowing her to choose between cheap and expensive transformations. This paragraph confirms the fundamental difference between Applicants' method and Collberg's method which is not intended to increase the complexity correlation between physical measures and characteristics of the program (as well as data) when the program is executed. For this reason this paragraph is not pertinent against Applicants' claims, and more particularly against claim 11, which recites a method for optimizing the choice of the next action to be taken by using an indirection table which contains the addresses of possible actions at an index calculated from variable items such as dynamic data or results from random draws. No such indirection table is alluded to in Collberg, either at the cited column 15, lines 26-43, or anywhere else. In addition Collberg, at column 15, lines 26-43, refers to choices (concerning the ordering of computations) which are to be made "at transformation time" and not at run time.

Claim 12 recites a method for introducing bypass codes having the characteristics of a particular sensitive calculation (such as a cryptographic computation), the aim being to mislead a potential attacker, deluding him into believing that the sensitive code is running when it is actually the bypass code. No such decoy mechanism is alluded to in Collberg. In contrast with the Examiner's assertion, Collberg only teaches in paragraph at column 12, lines 34 and 35 that it means for a program P' to be more obscure than a program P is a relatively vague appreciation because it must be based on human cognitive abilities. In the paragraph at column 18 lines 15-39, Collberg proposes to convert a section of code into a different virtual machine code. This new code is executed by a virtual machine interpreter included with the obfuscated application. Obviously, a particular application can contain several interpreters, each accepting a different language and executing a different section of the obfuscated application. This method does not correspond at all to the method as claimed in Applicants' claims 1 and 12.

In contrast with the Examiner's assertion, Applicants' claim 13 does not propose using of several interpreter each accepting different language: claim 13 proposes a step of introducing a plurality of implementation of certain instructions by enriching the set of instructions recognized by the interpreter.

Claims 14 to 17 are not suggested by Collberg. Collberg does not teach a step of introducing during the implementation of the instruction, a branching to a portion of one alternative code with a variable physical imprint or duration which dynamically determines the implementation to be executed. The paragraph at column 27, lines 20-33, cited with respect to claim 14, concerns the uses of opaque predicates that will require worst-case exponential time to break, on the combination of data races with interleaving and aliasing effects, such uses are without relation with the concept claimed in claim 14, which introduces a method to select the implementation to be executed, which does not appear in Collberg. Collberg, at column 27, lines 20-33, proposes a technique for managing data in an opaque way; it does not concern the selection of alternative implementations of instructions.

In addition, Collberg, at column 2, lines 40-57, cited against claim 15, presents options which are available or techniques which can be used during (or prior to) program transformations whereas claim 15 refers to implementation choices which are made at run time (and can thus differ for each execution of the program).

For at least these reasons, Applicants respectfully submit that claims 1-2 and 4-17 are patentable over the prior art of record. Applicants' claims 18 to 21 are believed to be patentable in and of themselves, and for the reasons discussed above with respect to claim 1.

In view of the above amendments and remarks, Applicant respectfully requests reconsideration and withdrawal of the outstanding rejections of record. Applicant submits that the application is in condition for allowance, and early notice to this effect is earnestly requested.

If the Examiner has any questions he is invited to contact the undersigned at 202-628-5197.

Respectfully submitted,

BROWDY AND NEIMARK, P.L.L.C.
Attorneys for Applicant

By /Ronni S. Jillions/
Ronni S. Jillions
Registration No. 31,979

RSJ:jmv
Telephone No.: (202) 628-5197
Facsimile No.: (202) 737-3528
G:\BN\M\Mout\HAMEAU2\pto\2007-03-30AMENDMENT.doc